From Double-Negation Translation of Proofs to Static Analysis of Code

Chuangjie Xu

SonarSource

Type Theory, Constructive Mathematics and Geometric Logic
1–5 May 2023, CIRM, Marseille, France

This talk is to relate

a technique for studying **proofs**

to

some tools for finding bugs in **software**

Gentzen's double-negation translation

Translating formulas in predicate logic as follows

$$
\begin{aligned}
(A \to B)^{\mathrm{G}} &:\equiv A^{\mathrm{G}} \to B^{\mathrm{G}} & P^{\mathrm{G}} &:\equiv \neg\neg P \quad \text{for primitive } P \\
(A \wedge B)^{\mathrm{G}} &:\equiv A^{\mathrm{G}} \wedge B^{\mathrm{G}} & (A \vee B)^{\mathrm{G}} &:\equiv \neg\neg (A^{\mathrm{G}} \vee B^{\mathrm{G}}) \\
(\forall x A)^{\mathrm{G}} &:\equiv \forall x A^{\mathrm{G}} & (\exists x A)^{\mathrm{G}} &:\equiv \neg\neg \exists x A^{\mathrm{G}}
\end{aligned}
$$

One can prove $\mathrm{CL} \vdash A \iff \mathrm{ML} \vdash A^{\mathrm{G}}$.

(The translations of Gentzen and Gödel are different in the sense of [Arthan&Oliva][1].)

[1] Rob Arthan and Paulo Oliva. **On affine logic and Lukasiewicz logic**. arXiv:1404.0570 [cs.LO], 2014.

Generalizing Gentzen's translation[2,3,4]

Replace ¬¬ by a **nucleus**, that is, an endofunction j on formulas such that

$$\phi \to j\phi \qquad (\phi \to j\psi) \to j\phi \to j\psi \qquad (j\phi)[t/x] \leftrightarrow j(\phi[t/x])$$

The translation becomes

$$P_j^{\mathrm{G}} := jP \qquad (\phi \to \psi)_j^{\mathrm{G}} := \phi_j^{\mathrm{G}} \to \psi_j^{\mathrm{G}}$$
$$(\phi \vee \psi)_j^{\mathrm{G}} := j(\phi_j^{\mathrm{G}} \vee \psi_j^{\mathrm{G}}) \qquad (\phi \wedge \psi)_j^{\mathrm{G}} := \phi_j^{\mathrm{G}} \wedge \psi_j^{\mathrm{G}}$$
$$(\exists x.\phi)_j^{\mathrm{G}} := j(\exists x.\phi_j^{\mathrm{G}}) \qquad (\forall x.\phi)_j^{\mathrm{G}} := \forall x.\phi_j^{\mathrm{G}}$$

Try e.g. $j\phi = (\phi \to X) \to X$ or $j\phi = \phi \vee \bot$

---

[2] Hajime Ishihara. **A Note on the Gödel–Gentzen Translation**. MLQ, 46(1):135–137, 2000.
[3] Martín Escardó and Paulo Oliva. **The Peirce translation**. APAL, 163(6):681–692, 2012.
[4] Benno van den Berg. **A Kuroda-style j-translation**. AML, 58(5-6):627–634, 2019.

Applying the Proofs-as-Programs correspondence

j-translation
of propositional logic + induction scheme

Proofs-as-Programs

parametrized syntactic translation
of Gödel's System T

Gödel's System T

Very briefly, $\mathsf{T} \equiv$ simply typed λ-calculus $+$ Nat $+$ primitive recursor

It can be given by

$$\begin{array}{lrcl} \textsf{Type} & \sigma, \tau & := & \mathsf{Nat} \mid \sigma \to \tau \mid \sigma \times \tau \mid \sigma + \tau \\ \textsf{Term} & t, u & := & x \mid \lambda x.t \mid tu \mid 0 \mid \mathrm{succ} \mid \mathrm{rec}_\sigma \mid \cdots \end{array}$$

with typing rules such as

$$\frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash tu : \tau} \qquad \frac{}{\Gamma \vdash \mathrm{rec}_\sigma : \sigma \to (\mathsf{Nat} \to \sigma \to \sigma) \to \mathsf{Nat} \to \sigma}$$

A Gentzen-style translation of System T[5]

A **nucleus** for System T is an endofunction J on T types together with T terms

$$\eta_\sigma : \sigma \to \mathrm{J}\sigma \qquad \kappa_{\sigma,\tau} : (\sigma \to \mathrm{J}\tau) \to \mathrm{J}\sigma \to \mathrm{J}\tau.$$

Given a nucleus (J, η, κ), we translate types $\sigma \mapsto \sigma^{\mathrm{J}}$ by

$$\mathsf{Nat}^{\mathrm{J}} := \mathrm{JNat} \qquad (\sigma \to \tau)^{\mathrm{J}} := \sigma^{\mathrm{J}} \to \tau^{\mathrm{J}}$$

$$(\sigma + \tau)^{\mathrm{J}} := \mathrm{J}(\sigma^{\mathrm{J}} + \tau^{\mathrm{J}}) \qquad (\sigma \times \tau)^{\mathrm{J}} := \sigma^{\mathrm{J}} \times \tau^{\mathrm{J}}$$

and terms $(t : \sigma) \mapsto (t^{\mathrm{J}} : \sigma^{\mathrm{J}})$ by

$$0^{\mathrm{J}} := \eta(0) \qquad (\lambda x.t)^{\mathrm{J}} := \lambda x^{\mathrm{J}}.t^{\mathrm{J}}$$

$$\mathrm{succ}^{\mathrm{J}} := \kappa(\eta \circ \mathrm{succ}) \qquad (tu)^{\mathrm{J}} := t^{\mathrm{J}}u^{\mathrm{J}} \qquad \ldots$$

[5] Chuangjie Xu. **A Gentzen-style monadic translation of Gödel's System T.** FSCD 2020.

Fundamental theorem of logical relation

(Omit $\sigma + \tau$ for simplicity: $\eta : \mathsf{Nat} \to \mathsf{JNat}$ $\kappa : (\mathsf{Nat} \to \mathsf{JNat}) \to \mathsf{JNat} \to \mathsf{JNat}$)

Let $[\![\_]\!]$ be the standard/natural interpretation of System T.

Extend a given $\mathrm{R}_{\mathsf{Nat}} \subseteq \mathbb{N} \times [\![\mathsf{JNat}]\!]$ to a logical relation $\mathrm{R}_\rho \subseteq [\![\rho]\!] \times [\![\rho^{\mathrm{J}}]\!]$ by

$$f\ \mathrm{R}_{\sigma\to\tau}\ g := \forall x^{[\![\sigma]\!]} a^{[\![\sigma^{\mathrm{J}}]\!]}\, (x\ \mathrm{R}_\sigma\ a \to fx\ \mathrm{R}_\tau\ ga)$$

Theorem. For any closed term $t : \rho$ in System T, we have

$$[\![t]\!]\ \mathrm{R}_\rho\ [\![t^{\mathrm{J}}]\!]$$

if $\forall n(n\ \mathrm{R}_{\mathsf{Nat}}\ [\![\eta]\!]n)$ and $\forall f^{\mathbb{N}\to\mathbb{N}} g^{\mathbb{N}\to[\![\mathsf{JNat}]\!]}\, (\forall i(fi\ \mathrm{R}_{\mathsf{Nat}}\ gi) \to f\ \mathrm{R}_{\mathsf{Nat}\to\mathsf{Nat}}\ [\![\kappa]\!]g)$.

Chuangjie Xu

Example: continuity of T-definable functions

Theorem. All functions $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ definable in System T are **continuous**, i.e.

$$\forall(\alpha : \mathbb{N}^{\mathbb{N}}).\exists(m : \mathbb{N}).\forall(\beta : \mathbb{N}^{\mathbb{N}}).\, (\alpha =_m \beta \to f\alpha = f\beta)$$

Proof. (1) Define a nucleus $\mathsf{JNat} = (\mathsf{Nat}^{\mathsf{Nat}} \to \mathsf{Nat}) \times (\mathsf{Nat}^{\mathsf{Nat}} \to \mathsf{Nat})$

$$\eta := \lambda n^{\mathsf{Nat}}.\langle \lambda\alpha.n\,;\lambda\alpha.0\rangle$$

$$\kappa := \lambda g^{\mathsf{Nat}\to\mathsf{JNat}} w^{\mathsf{JNat}}.\langle \lambda\alpha.(g(w_1\alpha))_1\alpha\,;\lambda\alpha.\max((g(w_1\alpha))_2\alpha, w_2\alpha)\rangle$$

(2) Define a relation $\mathrm{R}^{\alpha}_{\mathsf{Nat}} \subseteq \mathbb{N} \times [\![\mathsf{JNat}]\!]$

$$n\ \mathrm{R}^{\alpha}_{\mathsf{Nat}}\ (f, M) \ :=\ f\alpha = n \wedge \forall(\beta : \mathbb{N}^{\mathbb{N}}).(\alpha =_{M\alpha} \beta \to f\alpha = f\beta)$$

(3) Show that $\mathrm{R}^{\alpha}_{\mathsf{Nat}}$ preserves η and κ.

Chuangjie Xu

Other instances and applications

- Uniform continuity
- Totality
- Number of evaluation steps
- Majorizability[6]
- Closure under bar recursion[7,8]

[6] W. A. Howard. **Hereditarily majorizable functionals of finite type.** In Metamathematical Investigation of Intuitionistic Arithmetic and Analysis, pp. 454–461. Springer, Berlin, 1973.
[7] Helmut Schwichtenberg. **On bar recursion of types 0 and 1**. JSL, 44(3):325–329, 1979.
[8] Paulo Oliva and Silvia Steila. **A direct proof of Schwichtenberg's bar recursion closure theorem**. JSL, 83(1):70–83, 2018.

A Kuroda-style translation[9]

Given a nucleus (J, η, κ), we translate types $\sigma \mapsto \mathrm{J}[\sigma]$ where

$$[\mathsf{Nat}] := \mathsf{Nat} \qquad [\sigma \to \tau] := [\sigma] \to \mathrm{J}[\tau]$$
$$[\sigma + \tau] := [\sigma] + [\tau] \qquad [\sigma \times \tau] := [\sigma] \times [\tau]$$

And each term $t : \rho$ is translated to a term $[t] : \mathrm{J}[\rho]$.

How they differ — evaluation strategy

- The Gentzen-style translation is **call-by-name**.
- The Kuroda-style translation is **call-by-value**.

[9] Thomas Powell. **A functional interpretation with state**. LICS 2018.

Generalizing the syntactic translation to an interpreter

Parameters for the interpretation:

- A set **N** to model the base type Nat
- A monad J of sets

Types are recursively interpreted as in the translation:

$$[\![\mathsf{Nat}]\!]^{\mathrm{J}} := \mathrm{J}\mathbf{N} \qquad [\![\sigma \to \tau]\!]^{\mathrm{J}} := [\![\sigma]\!]^{\mathrm{J}} \to [\![\tau]\!]^{\mathrm{J}}$$

$$[\![\sigma + \tau]\!]^{\mathrm{J}} := \mathrm{J}([\![\sigma]\!]^{\mathrm{J}} + [\![\tau]\!]^{\mathrm{J}}) \qquad [\![\sigma \times \tau]\!]^{\mathrm{J}} := [\![\sigma]\!]^{\mathrm{J}} \times [\![\tau]\!]^{\mathrm{J}}$$

And each term $t : \sigma$ is interpreted by $[\![t]\!]^{\mathrm{J}} \in [\![\sigma]\!]^{\mathrm{J}}$.

Chuangjie Xu

Example: Escardó's dialogue trees[10]

The **dialogue monad** $\mathcal{D}$ for a given set X is inductively generated by

$$\eta : X \to \mathcal{D}X \qquad \beta : (\mathbb{N} \to \mathcal{D}X) \to \mathbb{N} \to \mathcal{D}X$$

Interpret System T by setting the parameters (**N**, J) to $(\mathbb{N}, \mathcal{D})$.

Recover the continuity result of System T definable functionals.

[10] Martín Escardó. **Continuity of Gödel's system T functionals via effectful forcing**. MFPS 2013.

Extending to sophisticated programming languages

To model additional language features,
the parameters should have corresponding structures:

- Base types / value constants – value-set parameters (e.g. **S** for strings)
- Primitive operations – e.g., concat: **J S** → **J S** → **J S**
- Conditional branching – join-semilattice
- Objects and fields – state monad
- Exceptions – option/either monad
- …

But the interpretation of the **functional core** is (essentially) the same.

A generic interpreter[11] of some typed language with strings

looks like the following

$$
\begin{aligned}
&\dots \\
[\![\lambda x.e]\!](\rho) &:= \lambda a.[\![e]\!](\rho, x \mapsto a) \\
[\![e_1 e_2]\!](\rho) &:= [\![e_1]\!](\rho)([\![e_2]\!](\rho)) \\
[\![e_1 + e_2]\!](\rho) &:= \mathrm{concat}([\![e_1]\!](\rho), [\![e_2]\!](\rho)) \\
[\![\textsf{if } e_1 \textsf{ then } e_2 \textsf{ else } e_3]\!](\rho) &:= \mathrm{join}([\![e_1]\!](\rho), [\![e_2]\!](\rho)) \\
&\dots
\end{aligned}
$$

where e.g. concat and join are additional assumptions on the monad $\mathtt{J}$.

Its instantiations can be used to find **bugs** of programs.

[11] Ulrich Schöpp and Chuangjie Xu. **A generic type system for featherweight Java**. FTfJP 2021.

What bugs are we interested in?

Example: SQL injection

```
String name = req.getParameter("name"); // source: user input
...
String query = "SELECT * FROM users WHERE name = '" + name + "'";
stmt.executeQuery(query); // sink: execute a tainted query
```

If the user input is `foo'; DROP TABLE users --` then

Vulnerability bugs where **tainted data** flows from **source** to **sink**

Idea of detecting such bugs

Consider the same example:

```
String name = req.getParameter("name"); // name:bad
...                                     // ...
String query = ... + name + "'";        // name:bad, query:bad
stmt.executeQuery(query);               // issue "query is bad"
```

- Track the taint information of all variables
- Raise an issue when a sink is applied to possibly tainted inputs

Taint analysis as an instance of the generic interpreter

- To **track taint information**

  interpret program expressions by the **taint lattice**, i.e.

$$\mathbf{S} = \mathcal{P}(\{\text{good}, \text{bad}\})$$

- To **raise issues**

  use the **writer monad**, i.e.

$$J(X) = X \times \text{String}$$

An abstract interpreter for taint analysis

Assume that the language has a taint source and a sink function, then

$$
\begin{aligned}
[\![\mathsf{source}]\!](\rho) &:= \text{return } \mathrm{bad} \\
[\![str]\!](\rho) &:= \text{return } \mathrm{good} \\
[\![\mathsf{sink}(e)]\!](\rho) &:= \text{do} \\
&\quad a \leftarrow [\![e]\!](\rho) \\
&\quad \text{when } (\mathrm{bad} \in a) \text{ tell } \text{“Boomm!”} \\
&\quad \text{return } ()
\end{aligned}
$$

E.g., for $(\_, s) = [\![e]\!] \in \mathcal{JS} = \mathcal{P}(\{\mathrm{good}, \mathrm{bad}\}) \times \mathrm{String}$

- if s is empty, then the program e is safe.

Summary

Double-negation translation	classical logic ↪ minimal logic
↓ parametrize	
j-translation of logic	classical logic ↪ intuitionistic logic ↪ minimal logic
↓ proofs-as-programs	
J-translation of System T	majorizability, continuity, bar recursion closure, …
↓ generalize	
Generic interpreter of System T	continuity via dialogue trees
↓ extend	
Generic interpreter of programming languages	static code analysis (to find bugs)